

**Law Office of
HOLLAND & KNIGHT LLP**

**701 Brickell Avenue
Suite 3000
Miami, Florida 33131
Telephone (305) 789-7773**

**Application
for
United States
Letters Patent**

filed on behalf of

**Applicant(s): Gheorghe C. Cascaval
Siddhartha Chatterjee**

**For: System and Method for
Encoding and Decoding Architecture
Registers
Attorney Docket: YOR920030127**

MIA1 #1248069 v1

PATENT

SYSTEM AND METHOD FOR ENCODING AND DECODING LARGE REGISTER FILES

CROSS-REFERENCE TO RELATED APPLICATIONS

5 [0001] Not Applicable.

STATEMENT REGARDING FEDERALLY SPONSORED-RESEARCH OR DEVELOPMENT

[0002] The research was sponsored by HPCS contract number NBCH020056
10 and ends September 16, 2003.

INCORPORATION BY REFERENCE OF MATERIAL SUBMITTED ON A COMPACT DISC

[0003] Not Applicable.
15

FIELD OF THE INVENTION

[0004] The invention disclosed broadly relates to the field of information
processing systems and more particularly relates to a system and method for
increasing the number of architecturally visible registers in a central processing unit
20 (CPU).

BACKGROUND OF THE INVENTION

[0005] Most current CPU architectures, including the PowerPC™ architecture, limit the number of architecturally visible registers (both general purpose registers, or GPRs, and floating point registers, or FPRs) to a small number N (with $N \leq 32$ in general for RISC architectures). While these architectural registers may be backed up by a larger pool of physical renaming registers, the compiler (or assembly language programmer) must make register allocation and spilling decisions using only (N-K) general-purpose registers, where K is the number of registers reserved for specific uses by the application binary interface. This limitation on the number of architected registers increases register pressure, increases the number of register spills and restores, and limits the use of program transformations requiring a large number of registers (such as unroll-and-jam).

[0006] The main architectural limitation on the size of the register files is the number of bits available in the instruction to encode register specifiers. Fixed length instructions and dense instruction encoding are key features of RISC architectures. For example, the PowerPC™ ISA (instruction set architecture) uses 32 bits to encode an instruction, with five bits allocated for each register specifier. In this architecture, instructions have between one and four register sources and destinations, such that there are only 12 bits remaining to encode the operation. Instruction encoding is tight, so that any attempt to increase the width of the register specifier fields would result either in longer instructions and code bloat, or in two-address instructions rather than the traditional RISC three-address instructions. We now discuss some known solutions

to the problem of encoding register specifiers to increase the number of usable registers.

Register windows in the Sparc™ architecture.

5 [0007] Several register contexts are maintained to improve the performance of
function call and return. However, while there may be more than N physical registers
present in the machine, only N of them are architecturally visible and available for use
at any point in time. The Zilog Z80 8-bit microprocessor had a primitive version of
this feature in the form of two sets of registers that could be rapidly exchanged with
10 one instruction.

Register renaming in super-scalar processors.

[0008] More than N registers are physically present in super-scalar processors,
but these registers are invisible to the user, being used by the hardware to transparently
15 and dynamically map the N architecturally visible registers over time. The user does
not have direct control over the management of this pool of registers.

Rotating register file.

[0009] The IA64 register file contains 128 architected registers encoded using
20 7 bit register specifier fields. This results in an instruction word that is 41 bits long,
resulting in reduced code density.

PATENT

The Rechtschaffen Patent.

[0010] United States Patent 4,574,349 (issued to Rechtschaffen) interprets the register specifiers in an instruction as indices into an indirection table from which the actual register numbers are obtained. This solution suffers from the following
5 drawbacks: (a) the indices to the indirection table are still limited to being log N bits wide -- this makes the solution similar to a user managed register renaming scheme; (b) they encode each register field independently, which requires either multiple indirection tables or multiple access ports into a shared indirection table, whereas we encode in a single entry of the indirection table all the register specifiers in an
10 instruction. A related solution is proposed in the ISCA '93 paper by Kiyohara et al., titled "Register Connections: A New Approach to Adding Registers into Instruction Set Architectures."

The CodePack™ System

15 [0011] The CodePack™ system is a hardware-software method for storing PowerPC™ instructions in memory in a compressed form and decompressing them on-the-fly, only as needed by the processor. This system relies on information-theoretic compression techniques to produce a variable-length encoding of PowerPC™ instructions that reduces the overall code footprint on average, but does not increase
20 the number of architected registers.

[0012] There is thus a need for a method and system for encoding and decoding architected registers that overcomes the shortcomings of the prior art.

SUMMARY OF THE INVENTION

[0013] A system and method to extend the number of architecturally visible registers in a processor while preserving the number of bits of the instruction encoding. The system comprises: an indirection table whose entries encode register patterns for registers used by instructions; instructions to load and store register patterns to and from the indirection table; a mechanism to identify instructions that use the indirection table; and a mechanism to identify a set of bits in instructions that are used to index into the indirection table.

10 [0014] According to another embodiment, a method of encoding registers in a computer instruction comprises constructing a table having a plurality of entries. Each entry specifies a combination of a plurality of registers. The method also comprises generating an instruction having a reference to one of the entries in the table. The method then comprises accessing the plurality of registers using the reference in the table. The method further comprises merging said number of registers into an expanded instruction that is used for remaining stages of instruction processing.

BRIEF DESCRIPTION OF THE DRAWINGS

20 [0015] FIG. 1 is an example of use of an indirection table (RAPT) by an instruction using a contiguous RAPT index field and merging in four register specifiers retrieved from the RAPT entry.

[0016] FIG. 2 shows the location of the RAPT in an instruction processing pipeline.

[0017] FIG. 3 is a flow chart illustrating a method according to the invention.

[0018] FIG. 4 is a simplified block diagram of an information processing
5 system comprising an embodiment of the invention.

DETAILED DESCRIPTION

[0019] The invention solves the problem of designing or enhancing the instruction set architecture (ISA) of a processor by increasing the number of
10 architected registers, while fulfilling the following requirements: a) continuing to encode instructions in the architected instruction width; b) preserving the three-address nature of instructions; and when applied to an existing ISA; c) maintaining backwards compatibility, to allow existing binaries to execute unmodified on implementations of the new ISA, while being able to access only the bottom N
15 registers (this requirement is vacuous when the invention is applied to a new ISA).

[0020] While in existing implementations of the PowerPC™ architecture 15 bits of the instruction are allocated for three register specifiers, programs do not use, and compilers do not generate, anywhere close to the 2^{15} possible combinations of
20 register specifiers. Thus, while the encoding of each register specifier is dense, the encoding of register specifier tuples is sparse. An ordered tuple of register specifiers occurring in a program is hereinafter referred to as a Register Access Pattern (RAP). Analysis of the SPECcpu2000 benchmarks shows 1000—3000 static RAPs per

benchmark. Of the thousands of RAPs, a small number account for most of the dynamic references made by a program.

[0021] Thus a system according to the invention uses RAPs to enhance a processor architecture and its corresponding ISA with four components: an architecturally visible Register Access Pattern Table (RAPT), which is used as an indirection table for RAPs; instructions to explicitly load and store RAPT entries; a method of identifying instructions that use the RAPT; and a set of bits in instructions that are used to index into the RAPT.

10

The RAPT

[0022] A RAPT entry comprises a plurality of register specifiers that together encode one or more RAPs. The number M of register specifiers in a RAPT entry is hereinafter called the RAPT width. For example, assuming a RAPT width $M = 4$ consider Table 1 below.

15

<i>RAPT index</i>	<i>RAPT entry</i>
1	1, 2, 3, 4
10	2, 3, 42, 69
31	128, 130, 141, 255

Table 1

[0023] The first column is the location of the RAP in the RAPT. The second column is the contents of the RAP. Thus, entry 1 of the RAPT contains the four numbers 1, 2, 3, and 4, which identify architectural register numbers to be used by an instruction accessing RAPT entry 1.

20

[0024] The number of addressable registers is determined by the width of each field of a RAPT entry, hereinafter called the implementation width. An implementation width of W bits allows the addressing of 2^W architectural registers. This limit is 256 for $W = 8$ (32-bit RAPT entries for $M = 4$) and 65536 for $W = 16$ (64-bit RAPT entries for $M = 4$). The number of bits used to index the RAPT is hereinafter denoted as B . The number of RAPT entries is therefore 2^B .

[0025] Referring to FIG. 1 there is shown an example of use of a RAPT by an instruction using a contiguous RAPT index field and merging in four register specifiers retrieved from the RAPT entry. An instruction 100 is processed by reading the RAPT index field 101. This field comprises an index that references entry j of the RAPT 102. The RAPT 102 is a table comprising four columns, each having a width W , and 2^B rows. In this example the register specifier field 101 comprises eight bits (i.e., $B = 8$). Those bits are used to identify RAPT entry j which comprises four register specifier segments each having seven bits, a total of 28 bits to characterize a register tuple. The resulting extended instruction 104 allows for many more architecturally visible registers being available for programming.

20 RAPT Management Instructions

[0026] We provide the following instructions to manage the RAPT: (a) An instruction that, given a RAPT index and a memory address, loads the contents of the memory address into the RAPT entry specified by the index; (b) An instruction that, given a RAPT index and a memory address, stores into said memory address the contents of the RAPT entry specified by the index; and (c) Optionally, instructions

may be provided for loading and storing all entries of the RAPT from/to a base memory address.

5

Identification of instruction mode

[0027] When applied to an existing ISA, the invention provides a mechanism, either explicit or implicit, to indicate for each instruction whether to interpret its register access fields directly (hereinafter referred to as compatibility mode), or using
10 indirection through the RAPT (hereinafter referred to as extended mode). A correctness criterion for this mechanism is that it must interpret each instruction of an existing binary as being in compatibility mode. This mechanism includes, but is not limited to, the following situations: always using the RAPT (implicitly); never using the RAPT (implicitly); specifying a mode on a per instruction basis (explicitly); and
15 specifying the mode of a group of instructions with a single mode context (the context is toggled explicitly, the classification of individual instructions is implicit).

Identification of RAPT index field

[0028] A system according to the invention requires the ISA to specify, for
20 each instruction, a plurality of B bits that are used to identify the RAPT entry to be used by an instance of that instruction (unconditionally, when the invention is applied to a new ISA; in extended mode, when the invention is applied to an existing ISA). These bits do not need to be contiguous in the instruction word. The collection of these bits is hereinafter called the RAPT index field of the instruction (e.g., FIG 1,
25 101).

Instruction processing

- [0029] Referring to FIG. 2, there is shown the location of the RAPT in an instruction processing pipeline. The enhanced architecture operates in the following manner: Instructions are always fetched (201) in the architected instruction width specified by the ISA. Register specifiers are then expanded to the implementation width W as instructions are decoded (202), following one of two possible paths. Either the instruction is executed in compatibility or extended mode. In the extended mode, the instruction is converted to an expanded instruction word.
- 10
- [0030] Referring to FIG. 3, there is shown a process 300 according to an embodiment of the invention. In step 302 an instruction is fetched from memory for processing. A decision 306 is then made to determine whether the fetched instruction is to be processed in compatibility mode or in extended mode. For an instruction in compatibility mode, in step 308 the register specifiers are extended to the implementation width W by padding with zeros to the left. Only registers 0 through (N-1) are accessible in this case. This path is relevant only in the case when the invention is applied to an existing ISA.
- 15
- [0031] For an instruction in extended mode, in step 310 the decoder extracts the RAPT index field of the instruction, uses it to index into the RAPT, and obtains the register specifiers from said RAPT entry. The specifiers are already represented at implementation width W.
- 20

[0032] In step 312 an appropriate number of extended register specifiers is merged with the remaining components of the instruction (i.e., after the register specifiers are extracted) and provided to the remainder of the instruction processing hardware. The actual number of register specifiers merged may vary from instruction to instruction; it is permissible for two instructions accessing the same RAPT entry to merge different numbers of register specifiers.

[0033] FIG. 4 is a simplified block diagram of a processor 400 according to an embodiment of the invention. The processor 400 comprises an input for fetching an ISA instruction 402 from memory. The processor 400 also comprises an indirection table (RAPT) 404 for storing register numbers as discussed in reference to FIG. 1. A set of instructions (RAPT Management Instructions) 406 for loading and storing RAPT entries is embedded in memory within the processor 400. A mechanism 408 identifies instructions that use the RAPT. A mechanism 410 is used to identify a set of bits in instructions that are used to index into the RAPT. The Instruction Mode Identification Mechanism 408 in conjunction with the RAPT are used to produce the extended instruction 412 (as discussed with respect to FIG. 1). The extended instruction is then processed (414) for merging a number of registers into an expanded instruction that is used for remaining stages of instruction processing.

20

[0034] What has been shown and discussed is a highly-simplified depiction of a programmable computer apparatus. Those skilled in the art will appreciate that other low-level components and connections are required in any practical application of a computer apparatus.

25

What is claimed is:

Express Mail No. *EV323493032US*

Docket No. YOR920030127